


GALAHAD ICFS

USER DOCUMENTATION GALAHAD Optimization Library version 5.1

1 SUMMARY

Given a symmetric matrix $\mathbf{A}$, this package **computes a symmetric, positive-definite approximation $\mathbf{L}\mathbf{L}^T$ using an incomplete Cholesky factorization**; the resulting matrix $\mathbf{L}$ is lower triangular. Subsequently, the solution $\mathbf{x}$ to the either of the linear systems $\mathbf{L}\mathbf{x} = \mathbf{b}$ and $\mathbf{L}^T\mathbf{x} = \mathbf{b}$ may be found for a given vector $\mathbf{b}$.

ATTRIBUTES — Versions: `GALAHAD_ICFS_single`, `GALAHAD_ICFS_double`. **Uses:** `GALAHAD_SYMBOLS`, `GALAHAD_SPECFILE` and `GALAHAD_SPACE`. **Date:** May 1998/December 2022. **Origin:** C.-J, Lin and J. J. Moré, Argonne National Laboratory, enhanced for modern fortran by N. I. M. Gould, Rutherford Appleton Laboratory. **Language:** Fortran 2003.

2 HOW TO USE THE PACKAGE

The package is available with single, double and (if available) quadruple precision reals, and either 32-bit or 64-bit integers. Access to the 32-bit integer, single precision version requires the `USE` statement

```
USE GALAHAD_ICFS_single
```

with the obvious substitution `GALAHAD_ICFS_double`, `GALAHAD_ICFS_quadruple`, `GALAHAD_ICFS_single_64`, `GALAHAD_ICFS_double_64` and `GALAHAD_ICFS_quadruple_64` for the other variants.

If it is required to use more than one of the modules at the same time, the derived types `ICFS_control_type`, `ICFS_inform_type`, `ICFS_data_type` and `NLPT_userdata_type`, (Section 2.2) and the subroutines `ICFS_initialize`, `ICFS_factorize`, `ICFS_solve`, `ICFS_terminate`, (Section 2.3) and `ICFS_read_specfile` (Section 2.5) must be renamed on one of the `USE` statements.

2.1 Real and integer kinds

We use the terms integer and real to refer to the fortran keywords `REAL(rp_)` and `INTEGER(ip_)`, where `rp_` and `ip_` are the relevant kind values for the real and integer types employed by the particular module in use. The former are equivalent to default `REAL` for the single precision versions, `DOUBLE PRECISION` for the double precision cases and quadruple-precision if 128-bit reals are available, and correspond to `rp_ = real32`, `rp_ = real64` and `rp_ = real128` respectively as defined by the fortran `iso_fortran_env` module. The latter are default (32-bit) and long (64-bit) integers, and correspond to `ip_ = int32` and `ip_ = int64`, respectively, again from the `iso_fortran_env` module.

2.2 The derived data types

Four derived data types are accessible from the package.

2.2.1 The derived data type for holding control parameters

The derived data type `ICFS_control_type` is used to hold controlling data. Default values may be obtained by calling `ICFS_initialize` (see Section 2.3.1), while components may also be changed by calling `GALAHAD_ICFS_read_spec` (see Section 2.5.1). The components of `ICFS_control_type` are:

error is a scalar variable of type INTEGER(ip_), that holds the stream number for error messages. Printing of error messages in ICFS_analyse, ICFS_estimate and ICFS_terminate is suppressed if error ≤ 0. The default is error = 6.

out is a scalar variable of type INTEGER(ip_), that holds the stream number for informational messages. Printing of informational messages in ICFS_analyse and ICFS_estimate is suppressed if out < 0. The default is out = 6.

print_level is a scalar variable of type INTEGER(ip_), that is used to control the amount of informational output which is required. No informational output will occur if print_level ≤ 0. If print_level > 01, details of any data errors encountered will be reported. The default is print_level = 0.

icfs_vectors is a scalar variable of type INTEGER(ip_), that holds the number of extra vectors of length n required by the incomplete Cholesky preconditioner. Usually, the larger the number, the better the preconditioner, but the more space and effort required to use it. Any negative value will be regarded as 0. The default is icfs_vectors = 10.

shift is a scalar variable of type REAL(rp_), that holds an initial estimate of the shift α used so that the incomplete factorization of $\mathbf{A}+\mathbf{D}$ is positive definite, where $\mathbf{D}$ is a diagonal matrix whose entries are no larger than α. This value α may subsequently be increased, as necessary, by the package, see inform%shift. The default is shift - 0.0.

space_critical is a scalar variable of type default LOGICAL, that must be set .TRUE. if space is critical when allocating arrays and .FALSE. otherwise. The package may run faster if space_critical is .FALSE. but at the possible expense of a larger storage requirement. The default is space_critical = .FALSE..

deallocate_error_fatal is a scalar variable of type default LOGICAL, that must be set .TRUE. if the user wishes to terminate execution if a deallocation fails, and .FALSE. if an attempt to continue will be made. The default is deallocate_error_fatal = .FALSE..

prefix is a scalar variable of type default CHARACTER and length 30, that may be used to provide a user-selected character string to preface every line of printed output. Specifically, each line of output will be prefaced by the string prefix(2:LEN(TRIM(prefix))-1), thus ignoreing the first and last non-null components of the supplied string. If the user does not want to preface lines by such a string, they may use the default prefix = "".

2.2.2 The derived data type for holding timing information

The derived data type ICFS_time_type is used to hold elapsed CPU and system clock times for the various parts of the calculation. The components of ICFS_time_type are:

total is a scalar variable of type REAL(rp_), that gives the total CPU time spent in the package.

factorize is a scalar variable of type REAL(rp_), that gives the CPU time spent factorizing the required matrices.

solve is a scalar variable of type REAL(rp_), that gives the CPU time spent solving the resulting triangular systems.

clock_total is a scalar variable of type REAL(rp_), that gives the total elapsed system clock time spent in the package.

clock_factorize is a scalar variable of type REAL(rp_), that gives the elapsed system clock time spent factorizing the required matrices.

clock_solve is a scalar variable of type REAL(rp_), that gives the elapsed system clock time spent solving the resulting triangular systems.

2.2.3 The derived data type for holding informational parameters

The derived data type ICFS_inform_type is used to hold parameters that give information about the progress and needs of the algorithm. The components of ICFS_inform_type are:

status is a scalar variable of type INTEGER(ip_), that gives the exit status of the algorithm. See Section 2.4 for details.

alloc_status is a scalar variable of type INTEGER(ip_), that gives the status of the last attempted array allocation or deallocation. This will be 0 if status = 0.

bad_alloc is a scalar variable of type default CHARACTER and length 80, that gives the name of the last internal array for which there were allocation or deallocation errors. This will be the null string if status = 0.

shift is a scalar variable of type REAL(rp_), that holds the final value of the shift α used so that the incomplete factorization of $\mathbf{A}+\mathbf{D}$ is positive definite, where $\mathbf{D}$ is a diagonal matrix whose entries are no larger than α.

time is a scalar variable of type ICFS_time_type whose components are used to hold elapsed CPU and system clock times for the various parts of the calculation (see Section 2.2.2).

2.2.4 The derived data type for holding problem data

The derived data type ICFS_data_type is used to hold all the data for a particular problem, or sequences of problems with the same structure, between calls of ICFS procedures. This data should be preserved, untouched, from the initial call to ICFS_initialize to the final call to ICFS_terminate.

2.3 Argument lists and calling sequences

There are four procedures for user calls (see Section 2.5 for further features):

1. The subroutine ICFS_initialize is used to set default values, and initialize private data.
2. The subroutine ICFS_factorize is called to form the incomplete Cholesky factor $\mathbf{L}$ from $\mathbf{A}$.
3. The subroutine ICFS_triangular_solve is called to solve either of the triangular systems $\mathbf{L}\mathbf{x} = \mathbf{b}$ or $\mathbf{L}^T\mathbf{x} = \mathbf{b}$ for given vectors $\mathbf{b}$.
4. The subroutine ICFS_terminate is provided to allow the user to automatically deallocate array components of the private data, allocated by ICFS_factorize, at the end of the solution process. It is important to do this if the data object is re-used for another matrix **with a different structure** since ICFS_initialize cannot test for this situation, and any existing associated targets will subsequently become unreachable.

We note that in addition to the above calls, the user may also call the original fortran 77 subroutines DICFS and DSTRSOL directly. For details of the necessary argument lists, see Section 6.

2.3.1 The initialization subroutine

Default values are provided as follows:

```
CALL ICFS_initialize( data, control, inform )
```

data is a scalar INTENT(INOUT) argument of type ICFS_data_type (see Section 2.2.4). It is used to hold data about the matrix and its factors.

control is a scalar INTENT(OUT) argument of type ICFS_control_type (see Section 2.2.1). On exit, control contains default values for the components as described in Section 2.2.1. These values should only be changed after calling ICFS_initialize.

inform is a scalar INTENT(OUT) argument of type ICFS_inform_type (see Section 2.2.3). A successful call to ICFS_initialize is indicated when the component status has the value 0. For other return values of status, see Section 2.4.

2.3.2 The incomplete factorization subroutine

The factorization phase, in which incomplete Cholesky factors **L** of **A** are determined, is performed as follows:

```
CALL ICFS_factorize( n, PTR, ROW, DIAG, VAL, data, control, inform )
```

n is a scalar INTENT(IN) scalar argument of type INTEGER(ip_), that must be set to n the dimension of the matrix **A**. **Restrictions:** n > 0.

PTR is a scalar INTENT(IN) rank-one array argument of type INTEGER(ip_) and dimension n + 1, whose j-th component gives the starting address for list of nonzero values and their corresponding row indices in column j of the **strict lower triangular part** of **A** (The entry $a_{i,j}$ is in the strict lower triangular part of **A** if $i > j$). That is, the nonzeros in column j of the strict lower triangle of **A** must be in positions PTR(j) ..., PTR(j+1) - 1 for j = 1, ...n. Note that PTR(n+1) points to the first position beyond that needed to store **A**.

ROW is a scalar INTENT(IN) rank-one array argument of type INTEGER(ip_) and dimension at least PTR(n+1)-1, that contains the row indices of the strict lower triangular part of **A** in the compressed column storage format described above.

DIAG is a scalar INTENT(IN) rank-one array argument of type REAL(rp_) and dimension n, whose j-th component contains the value of the j-th diagonal of **A**.

VAL is a scalar INTENT(IN) rank-one array argument of type REAL(rp_) and dimension at least PTR(n+1)-1, that contains the values of the entries strict lower triangular part of **A** that correspond to the row indices described above.

data is a scalar INTENT(INOUT) argument of type ICFS_data_type (see Section 2.2.4). It is used to hold data about the problem being solved. It must not have been altered **by the user** since the last call to ICFS_initialize.

control is a scalar INTENT(IN) argument of type ICFS_control_type (see Section 2.2.1). Default values may be assigned by calling ICFS_initialize prior to the first call to ICFS_analyse.

inform is a scalar INTENT(INOUT) argument of type ICFS_inform_type (see Section 2.2.3). A successful call to ICFS_analyse is indicated when the component status has the value 0. For other return values of status, see Section 2.4.

2.3.3 The triangular solution subroutine

The solution phase, in which one of the triangular systems $\mathbf{L}\mathbf{x} = \mathbf{b}$ or $\mathbf{L}^T\mathbf{x} = \mathbf{b}$ for given vectors **b**, is performed as follows:

```
CALL ICFS_triangular_solve( n, X, transpose, data, control, inform )
```

n, data, control and inform are exactly as described and input to ICFS_factorize, and must not have been changed in the interim.

X is a scalar INTENT(INOUT) rank-one array argument of type REAL(rp_), and dimension n, that must be set on input so that X(i) contains the component b_i, $i = 1, \ldots,$ n of the vector **b**. On output, this will have been overwritten by the desired vector **x**.

transpose is a scalar INTENT(IN) argument of type default LOGICAL, that must be set .FALSE. if the user wishes to solve $\mathbf{L}\mathbf{x} = \mathbf{b}$ and .TRUE. if instead the solution to $\mathbf{L}^T\mathbf{x} = \mathbf{b}$ is sought.

2.3.4 The termination subroutine

All previously allocated arrays are deallocated as follows:

```
CALL ICFS_terminate( data, control, inform )
```

data is a scalar INTENT(INOUT) argument of type ICFS_data_type exactly as for ICFS_factorize, which must not have been altered **by the user** since the last call to ICFS_initialize. On exit, array components will have been deallocated.

control is a scalar INTENT(IN) argument of type ICFS_control_type exactly as for ICFS_factorize.

inform is a scalar INTENT(OUT) argument of type ICFS_inform_type exactly as for ICFS_factorize. Only the component status will be set on exit, and a successful call to ICFS_terminate is indicated when this component status has the value 0. For other return values of status, see Section 2.4.

2.4 Warning and error messages

A negative value of inform%status on exit from ICFS_factorize, ICFS_triangular_solve or ICFS_terminate indicates that an error has occurred. No further calls should be made until the error has been corrected. Possible values are:

-1. An allocation error occurred. A message indicating the offending array is written on unit control%error, and the returned allocation status and a string containing the name of the offending array are held in inform%alloc_status and inform%bad_alloc, respectively.

-2. A deallocation error occurred. A message indicating the offending array is written on unit control%error and the returned allocation status and a string containing the name of the offending array are held in inform%alloc_status and inform%bad_alloc, respectively.

-3. The restriction $0 <$ n has been violated.

2.5 Further features

In this section, we describe an alternative means of setting control parameters, that is components of the variable control of type ICFS_control_type (see Section 2.2.1), by reading an appropriate data specification file using the subroutine ICFS_read_specfile. This facility is useful as it allows a user to change ICFS control parameters without editing and recompiling programs that call ICFS.

A specification file, or specfile, is a data file containing a number of "specification commands". Each command occurs on a separate line, and comprises a "keyword", which is a string (in a close-to-natural language) used to identify a control parameter, and an (optional) "value", which defines the value to be assigned to the given control parameter. All keywords and values are case insensitive, keywords may be preceded by one or more blanks but values must not contain blanks, and each value must be separated from its keyword by at least one blank. Values must not contain more than 30 characters, and each line of the specfile is limited to 80 characters, including the blanks separating keyword and value.

The portion of the specification file used by ICFS_read_specfile must start with a "BEGIN ICFS" command and end with an "END" command. The syntax of the specfile is thus defined as follows:

```
( .. lines ignored by ICFS_read_specfile .. )
  BEGIN ICFS
     keyword    value
     .......    .....
     keyword    value
  END
( .. lines ignored by ICFS_read_specfile .. )
```

where keyword and value are two strings separated by (at least) one blank. The "BEGIN ICFS" and "END" delimiter command lines may contain additional (trailing) strings so long as such strings are separated by one or more blanks, so that lines such as

```
BEGIN ICFS SPECIFICATION
```

and

```
END ICFS SPECIFICATION
```

are acceptable. Furthermore, between the "BEGIN ICFS" and "END" delimiters, specification commands may occur in any order. Blank lines and lines whose first non-blank character is ! or * are ignored. The content of a line after a ! or * character is also ignored (as is the ! or * character itself). This provides an easy manner to "comment out" some specification commands, or to comment specific values of certain control parameters.

The value of a control parameters may be of three different types, namely integer, logical or real. Integer and real values may be expressed in any relevant Fortran integer and floating-point formats (respectively). Permitted values for logical parameters are "ON", "TRUE", ".TRUE.", "T", "YES", "Y", or "OFF", "NO", "N", "FALSE", ".FALSE." and "F". Empty values are also allowed for logical control parameters, and are interpreted as "TRUE".

The specification file must be open for input when ICFS_read_specfile is called, and the associated device number passed to the routine in device (see below). Note that the corresponding file is REWINDed, which makes it possible to combine the specifications for more than one program/routine. For the same reason, the file is not closed by ICFS_read_specfile.

2.5.1 To read control parameters from a specification file

Control parameters may be read from a file as follows:

```
CALL ICFS_read_specfile( control, device )
```

control is a scalar INTENT(INOUT) argument of type ICFS_control_type (see Section 2.2.1). Default values should have already been set, perhaps by calling ICFS_initialize. On exit, individual components of control may have been changed according to the commands found in the specfile. Specfile commands and the component (see Section 2.2.1) of control that each affects are given in Table 2.1.

device is a scalar INTENT(IN) argument of type INTEGER(ip_), that must be set to the unit number on which the specfile has been opened. If device is not open, control will not be altered and execution will continue, but an error message will be printed on unit control%error.

2.6 Information printed

If control%print_level is positive, information about errors encountered will be printed on unit control%out.

command	component of control	value type
error-printout-device	%error	integer
printout-device	%out	integer
print-level	%print_level	integer
number-of-icfs-vectors	icfs_vectors	integer
initial-shift	shift	real
space-critical	%space_critical	logical
deallocate-error-fatal	%deallocate_error_fatal	logical
output-line-prefix	%prefix	character

Table 2.1: Specfile commands and associated components of control.

3 GENERAL INFORMATION

Use of common: None.

Workspace: Provided automatically by the module.

Other routines called directly: None.

Other modules used directly: The module uses the GALAHAD packages GALAHAD_SYMBOLS, GALAHAD_SPECFILE and GALAHAD_SPACE.

Input/output: Output is under control of the arguments control%error, control%out and control%print_level.

Restrictions: n $>$ 0 .

Portability: ISO Fortran 2003. The package is thread-safe.

4 METHOD

The package computes incomplete Cholesky factors **L** of **A** so that

$$\mathbf{A} + \mathbf{D} = \mathbf{L}\mathbf{L} + \mathbf{E}.$$

The pattern of entries in **L** match those in **A** with additional "fill-ins" controlled by the extra storage provided. The shifted diagonal **D** whose values do not exceed a scalar shift α allows for indefinite **A**, and also guarantees stable factors **L**. Increasing the extra storage provided generally decreases the size of the error matrix **E**, but increases the cost of the algorithm.

Reference:

The method is described in detail in

C.J. Lin and J. J. Moreé. "Incomplete Cholesky factorizations with limited memory". SIAM J. Sci. Computing. **21** (1999) 24–45.

5 EXAMPLES OF USE

Suppose that

$$\mathbf{A} = \begin{pmatrix} 2 & 1 & & & \\ 1 & 5 & 1 & & 1 \\ & 1 & 1 & 1 & \\ & & 1 & 7 & \\ & 1 & & & 2 \end{pmatrix} \text{ and } \mathbf{b} = \begin{pmatrix} 3 \\ 8 \\ 3 \\ 8 \\ 3 \end{pmatrix},$$

where the missing entries in $\mathbf{A}$ are structural zeros. Then we may find suitable incomplete factors $\mathbf{L}$, and subsequently solve $\mathbf{L}\mathbf{y} = \mathbf{b}$ and $\mathbf{L}^T\mathbf{x} = \mathbf{y}$ using the following code; notice that we initialize $\mathbf{b}$ in $\mathbf{x}$, overwrite this with $\mathbf{y}$ in the first triangular solve, and finally recover $\mathbf{x}$ from $\mathbf{y}$ in the second solve:

```
   PROGRAM ICFS_EXAMPLE   !  GALAHAD 4.1 - 2022-12-04 AT 09:30 GMT.
   USE GALAHAD_ICFS_double
   IMPLICIT NONE
   INTEGER, PARAMETER :: wp = KIND( 1.0D+0 )
   TYPE ( ICFS_data_type ) :: data
   TYPE ( ICFS_control_type ) control
   TYPE ( ICFS_inform_type ) :: inform
   INTEGER, PARAMETER :: n = 5
   INTEGER, PARAMETER :: ne = 5
   INTEGER, ALLOCATABLE, DIMENSION( : ) :: PTR, ROW
   REAL ( KIND = wp ), ALLOCATABLE, DIMENSION( : )  :: DIAG, VAL
   REAL ( KIND = wp ) :: X( n )
! allocate and set lower triangle of matrix in sparse by column form
   ALLOCATE( PTR( n + 1 ), DIAG( n ), VAL( ne ), ROW( ne ) )
   PTR = (/ 1, 2, 4, 5, 5, 5 /)
   ROW = (/ 2, 3, 5, 4 /)
   DIAG = (/ 2.0_wp, 5.0_wp, 1.0_wp, 7.0_wp, 2.0_wp /)
   VAL = (/ 1.0_wp, 1.0_wp, 1.0_wp, 1.0_wp /)
! problem setup complete
   CALL ICFS_initialize( data, control, inform )
   control%icfs_vectors = 1
! form and factorize the preconditioner, P = L L^T
   CALL ICFS_factorize( n, PTR, ROW, DIAG, VAL, data, control, inform )
   IF ( inform%status < 0 ) THEN
     WRITE( 6, '( A, I0 )' )                                                &
          ' Failure of ICFS_factorize with status = ', inform%status
     STOP
   END IF
! use the factors to solve L L^T x = b, with b input in x
   X( : n ) = (/ 3.0_wp, 8.0_wp, 3.0_wp, 8.0_wp, 3.0_wp /)
   CALL ICFS_triangular_solve( n, X, .FALSE., data, control, inform )
   CALL ICFS_triangular_solve( n, X, .TRUE., data, control, inform )
   IF ( inform%status == 0 ) THEN
     WRITE( 6, "( ' ICFS - Preconditioned solution is ', 5F6.2 )" ) X
   ELSE
     WRITE( 6, "( ' ICFS - exit status = ', I0 )" ) inform%status
   END IF
! clean up
   CALL ICFS_terminate( data, control, inform )
   DEALLOCATE( DIAG, VAL, ROW, PTR )
   STOP
   END PROGRAM ICFS_EXAMPLE
```

The code produces the following output:

```
 ICFS - Preconditioned solution is   1.00  1.00  1.00  1.00  1.00
```

6 APPENDIX

The original subroutine calls DICFS and DSTRSOL, as encoded in ICFS_factorize and ICFS_triangular_solve, may also be called as part of the package. To quote from the package source,

```
!     Subroutine dicfs
!
!     Given a symmetric matrix A in compressed column storage, this
!     subroutine computes an incomplete Cholesky factor of A + alpha*D,
!     where alpha is a shift and D is the diagonal matrix with entries
!     set to the l2 norms of the columns of A.
!
!     The subroutine statement is
!
!       subroutine dicfs(n,nnz,a,adiag,acol_ptr,arow_ind,l,ldiag,lcol_ptr,
!                        lrow_ind,p,alpha,iwa,wa1,wa2)
!
!     where
!
!       n is an integer variable.
!         On entry n is the order of A.
!         On exit n is unchanged.
!
!       nnz is an integer variable.
!         On entry nnz is the number of nonzeros in the strict lower
!            triangular part of A.
!         On exit nnz is unchanged.
!
!       a is a real array of dimension nnz.
!         On entry a must contain the strict lower triangular part
!            of A in compressed column storage.
!         On exit a is unchanged.
!
!       adiag is a real array of dimension n.
!         On entry adiag must contain the diagonal elements of A.
!         On exit adiag is unchanged.
!
!       acol_ptr is an integer array of dimension n + 1.
!         On entry acol_ptr must contain pointers to the columns of A.
!            The nonzeros in column j of A must be in positions
!            acol_ptr(j), ... , acol_ptr(j+1) - 1.
!         On exit acol_ptr is unchanged.
!
!       arow_ind is an integer array of dimension nnz.
!         On entry arow_ind must contain row indices for the strict
!            lower triangular part of A in compressed column storage.
```

```
!         On exit arow_ind is unchanged.
!
!       l is a real array of dimension nnz+n*p.
!         On entry l need not be specified.
!         On exit l contains the strict lower triangular part
!            of L in compressed column storage.
!
!       ldiag is a real array of dimension n.
!         On entry ldiag need not be specified.
!         On exit ldiag contains the diagonal elements of L.
!
!       lcol_ptr is an integer array of dimension n + 1.
!         On entry lcol_ptr need not be specified.
!         On exit lcol_ptr contains pointers to the columns of L.
!            The nonzeros in column j of L are in the
!            lcol_ptr(j), ... , lcol_ptr(j+1) - 1 positions of l.
!
!       lrow_ind is an integer array of dimension nnz+n*p.
!         On entry lrow_ind need not be specified.
!         On exit lrow_ind contains row indices for the strict lower
!            triangular part of L in compressed column storage.
!
!       p is an integer variable.
!         On entry p specifes the amount of memory available for the
!            incomplete Cholesky factorization.
!         On exit p is unchanged.
!
!       alpha is a real variable.
!         On entry alpha is the initial guess of the shift.
!         On exit alpha is final shift
!
!       iwa is an integer work array of dimension 3*n.
!
!       wa1 is a real work array of dimension n.
!
!       wa2 is a real work array of dimension n.
!
!     Subroutine dstrsol
!
!     This subroutine solves the triangular systems L*x = r or L'*x = r.
!
!     The subroutine statement is
!
!       subroutine dstrsol(n,l,ldiag,jptr,indr,r,task)
!
!     where
!
!       n is an integer variable.
!         On entry n is the order of L.
!         On exit n is unchanged.
!
```

```
!       l is a real array of dimension *.
!         On entry l must contain the nonzeros in the strict lower
!            triangular part of L in compressed column storage.
!         On exit l is unchanged.
!
!       ldiag is a real array of dimension n.
!         On entry ldiag must contain the diagonal elements of L.
!         On exit ldiag is unchanged.
!
!       jptr is an integer array of dimension n + 1.
!         On entry jptr must contain pointers to the columns of A.
!            The nonzeros in column j of A must be in positions
!            jptr(j), ... , jptr(j+1) - 1.
!         On exit jptr is unchanged.
!
!       indr is an integer array of dimension *.
!         On entry indr must contain row indices for the strict
!            lower triangular part of L in compressed column storage.
!         On exit indr is unchanged.
!
!       r is a real array of dimension n.
!         On entry r must contain the vector r.
!         On exit r contains the solution vector x.
!
!       task is a character variable of length 60.
!         On entry
!            task(1:1) = 'N' if we need to solve L*x = r
!            task(1:1) = 'T' if we need to solve L'*x = r
!         On exit task is unchanged.
!
!     MINPACK-2 Project. October 1998.
!     Argonne National Laboratory.
!     Chih-Jen Lin and Jorge J. More'.
```